CLEAN VERSION

In The Specification:

Paragraph beginning at line 26 of page 2 has been amended as follows:

FIG. 9A is a UML diagram of a preferred embodiment for the Forest and Edge classes; and

FIG. 9B is a UML diagram of a preferred embodiment for the Tree and TreeNode classes.

Paragraph beginning at line 3 of page 3 has been amended as follows:

Referring to **FIG. 1**, a schematic diagram of a system 10 for encoding and decoding multiple parses is shown. The speech recognizer (SR) **11** produces output or word graph **12**, which is sent to parser **14**. A grammar **16** is used to produce a packed representation **18** of the multiple parses. A preferred method and system for encoding the packed representation **18** is disclosed in U.S. Application No. 09/054,601 entitled "Method, Device and System For Generalized Bidirectional Island-Driven Chart Parsing," now issued as U.S. Patent Number 6,128,596, the entire disclosure of which is incorporated herein by reference. The packed representation **18** is then unpacked using the unpacking module **20** to create an unpacked forest **22** which is input to a semantic interpretation module **24**. A flow chart of a preferred

embodiment of the unpacking module **20** is presented in **FIGS. 2** and **3A**. The unpacking module **20** may include any suitable computer programming code using any conventional programming language, and may be stored on any conventional computer readable medium.

Paragraph beginning at line 8 of page 4 has been amended as follows:

The parser's packed representation **25** (see **FIG. 5**) consists of a hierarchically ordered set of edgenodes corresponding to complete edges from the parser's chart. The "most probable" parse tree that is returned by the parser **14** (and which provides the basis for further processing) is constructed by extracting from the packed representation **25**, the information necessary to construct a treenode corresponding to the most probable edgenode at each node of the parse tree. The edge class is used by the parser for encoding the course of the parse; consequently, edge objects are relatively large and have complex behavior. Although encoding a parse tree by use of edge objects has no apparent impact on parser performance when it returns only one parse tree per input word graph, returning all possible parses in this manner may have an adverse impact on system performance. A solution to this problem is to use the C++ inheritance mechanism to split the data and behavior of the edge class into: (1) a base class ("treenode"), which contains only the data and behavior that is necessary to define and process parse trees returned by the parser, and (2) a derived class (treenode: edge), which contains all of the remaining data and behavior (required for constructing and maintaining the parses in the parser's agenda and chart objects). In the discussion that follows, the base class "treenode" will be referred to as a treenode, while the derived class "edge" will be referred to as an edgenode.

Paragraph beginning at line 26 of page 5 has been amended as follows:

B4

An unordered set of alternative parses can be obtained from a given packed representation **25** by an exhaustive traversal of the representation which incorporates the processing of the substitution lists as described below. If $\{E_n\}$ is the set of n edgenodes that define the most probable parse obtained from the packed representation, and k_i is the number of edgenodes in the substitution list corresponding to the edgenode E_i, then the number of parse trees returned by the traversal will be less than or equal to $\prod_{i=1}^{n} k_i$. The number of parse trees may be less than this number due to dependencies resulting from the dominance relationships in the tree; (e.g. in the example shown in **FIG. 5**, there are only three parses (rather than four) because the substitution list containing edgenodes D0 and D1 is dominated by only one of the edgenodes in the substitution list containing edgenodes C0 and C1, not both of them).

Paragraph beginning at line 1 of page 15 has been amended as follows:

B5

The next edgenode is now obtained (block 59) as shown in **FIG. 3A**. Because the current edgenode **30D0** has at least one child (block 66), the next edgenode is set to the leftmost child of the current edgenode (block 68), that is, to edgenode **3OW2**. Because the next edgenode is not NULL (block 60), the current edgenode is set to the next edgenode **30W2** (block 61).

Paragraph beginning at line 6 of page 15 has been amended as follows:

Next, because the current edgenode **30W2** is the first edgenode in its substitution list **32W2** (block 52), N is set to the length of the substitution list for the current edgenode (block 53). For edgenode **30W2**, the length of the substitution list **32W2** is equal to 1. Because N is not greater than 1 (block 54) a scalar update of the current forest F0.0.0 is performed with the current edgenode (block 55), resulting in a treenode labeled w2 being added to the current forest F0.0.0, under its active node (the treenode labeled D0). Because the edgenode **30W2** is a terminal node, its corresponding treenode labeled w2 is also a terminal node, thus the treenode labeled D0 remains the active node in the current forest F0.0.0 (95).

Paragraph beginning at line 4 of page 19 has been amended as follows:

The next edgenode is now obtained (block 59) as shown in **FIG. 3A**. Because the current edgenode **30W5** does not have at least one child (block 66), it is tested to see if it is the last edgenode in the substitution list **32W5** (block 70). Because it is the only edgenode in the substitution list **32W5**, it is necessarily the last one, so it is tested to see if it has a sibling to its immediate right (block 76). Because it does not have a sibling to its immediate right, the node closure operation is applied to the current forest F0.0 (block 77). Because the current forest F0.0 contains the two forests F0.0.0 and F0.0.1, the application of the node closure operation to it results in the node closure operation being applied to each of these contained forests. The active node of the forest F0.0.0 is the treenode labeled C0. Thus the application of the node closure

operation to the forest F0.0.0 results in the treenode labeled A (which is the parent node of the treenode labeled C0) becoming the new active node of the forest F0.0.0. Similarly, the active node of the forest F0.0.1 is the treenode labeled C0. Thus the application of the node closure operation to the forest F0.0.1 results in the treenode labeled A (which is the parent node of the treenode labeled C0) becoming the new active node of the forest F0.0.1.

Paragraph beginning at line 1 of page 23 has been amended as follows:

The next edgenode is now obtained (block 59) as shown in **FIG. 3A**. Because the current edgenode **30W6** does not have at least one child (block 66), it is tested to see if it is the last edgenode in the substitution list **32W6** (block 70). Because it is the only edgenode in the substitution list **32W6**, it is necessarily the last one, so it is tested to see if it has a sibling to its immediate right (block 76). Because it does not have a sibling to its immediate right, the node closure operation is applied to the current forest F0.1, resulting in the parent treenode of the treenode labeled E (i.e., the treenode labeled C1) becoming the new active node of the current forest F0.1 (block 77). The current edgenode is then tested to see if it has a parent (block 82). Because the current edgenode **30W6** has a parent (edgenode **30E**), the current edgenode becomes edgenode **30E** (block 86). Because the current forest is F0.1, not F0 (block 87), it is changed to the forest of the parent of the current edgenode (block 88), that is, to F0.1 (thus remaining unchanged).